UCRL-WEB-228716

Moab at LC

Table of Contents

Preface

Scope: This manual explains the role of the Moab Workload Manager in controlling and scheduling batch jobs on LC production computers. The relationship between LCRM (URL: http://www.llnl.gov/LCdocs/dpcs) (the Livermore Computing Resource Manager and Moab's predecessor) and Moab (including user tools, job-control options, and job-status information), and between SLURM (URL: http://www.llnl.gov/LCdocs/slurm) (the Simple Linux Utility for Resource Management) and Moab, both receive special attention and explicit comparison. LC started to use Moab to gradually replace LCRM, working on top of SLURM, early in 2007.

Availability: Moab will be introduced on each new OCF production computer starting with ATLAS.

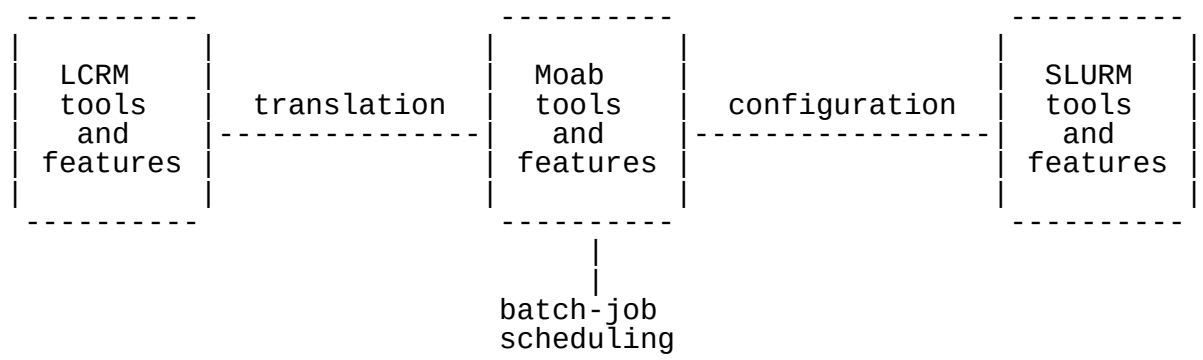
Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at:

```
OCF: http://www.llnl.gov/LCdocs/moab/moab.pdf
SCF: http://www.llnl.gov/LCdocs/moab/moab_scf.pdf
```

Introduction

Starting in early 2007, LC began gradually replacing its locally developed batch-job ("workload") management system (called "Livermore Computing Resource Manager" or LCRM (URL: http://www.llnl.gov/LCdocs/dpcs)) with a commercial product from Cluster Resources, Inc. (URL: http://www.clusterresources.com) called "Moab Workload Manager" (sometimes MWM but more often just Moab). Moab fills the across-cluster job-scheduling, tracking, and reporting role that LCRM formerly filled, while coordinating with SLURM (URL: http://www.llnl.gov/LCdocs/slurm), which actually manages nodes and allocates computing resources to jobs when they run on any specific LC cluster.



- LCRM-Moab Translation.
  As installed at LC, Moab is configured to transparently replace LCRM features with few user changes needed. Those who run batch jobs can still use LCRM tools (such as PSUB to submit jobs), job-control scripts (with imbedded #PSUB instructions), and job-monitoring reports (such as generated by PSTAT), which Moab converts as needed automatically. Native Moab commands and features are also available for those who prefer them. For more details on this LCRM-Moab conversion, see the LCRM-Moab Interaction section (page 5) below.
- SLURM-Moab Interaction.
  Moab and SLURM are both configured on LC machines to gracefully interact and smoothly coordinate across-cluster (e.g., domain-wide limit) and within-cluster (e.g., node or CPU assignment) job-management duties. For more details on this coordination, see the SLURM-Moab Interaction section (page 24) below.
- Scheduling.
  Moab fills, monitors, controls, and throttles an across-cluster queue of submitted batch jobs, just as did LCRM. For a closer look at the job-scheduling policies that Moab uses, and how privileges and exemptions work, see the Job Scheduling section (page 25) below.
- Remaining Time.
  LCRM offered two ways (with signals and polling) for a job to detect its remaining time and manage it for graceful termination. Several new alternatives are available under Moab (and SLURM). See the Remaining Time Alternatives section (page 26) for a comparative explanation of each strategy.

LCRM-Moab Translation

This section compares LCRM job-control features and tools with their Moab counterparts. It also discusses when and how LC automatically translates from LCRM to Moab for situations where such conversion is possible.

Tool Comparison

On LC machines where Moab has replaced LCRM, job-control tools fall into two broad groups:

- those former LCRM tools (such as PSUB) that are *emulated* by Moab (that is, they have been replaced with wrapper scripts that accept most former LCRM-style execute lines and *automatically* convert them to corresponding native Moab commands and options with no extra work on your part), and
- those tools (such as PSHARE) that you must abandon in favor of invoking new Moab counterparts (such as MDIAG) explicitly, usually with different options or combinations of options to perform the old functions.

Some old LCRM tools have no specific Moab counterparts (perhaps none are needed), and of course some new Moab tools (for example, SHOWSTATS) fill roles not addressed under LCRM. The table below lists job-control tools by function (or role) and indicates which are emulated, which you must personally replace with a Moab alternative, and which have no replacment (yet). Separate sections below spell out for each tool's *options* that option's translation or substitution under Moab (if any).

All emulation wrapper scripts for LCRM tools, as well as all native Moab job-control tools, reside in /usr/bin on those LC machines where Moab has been installed.

Function	**Original LCRM Tool**	**Emulated by Moab?**	**Similar Moab Tool**
Six core LCRM tools:			
Submit a job	PSUB	yes	MSUB
Change a submitted job	PALTER	yes	MJOBCTL -m
Monitor submitted job(s)	PSTAT -A PSTAT -f	yes	SHOWQ CHECKJOB
Hold a sumbitted job	PHOLD	yes	MJOBCTL -h
Release a held job	PREL	yes	MJOBCTL -u
Remove a submitted job	PRM	yes	MJOBCTL -c
Other job-control tools:			
Report bank names, privileges	BAC -u	no	MDIAG -u
Change default bank	DEFBANK -u	no	MDIAG -u
Expedite a job	PEXP	no	MJOBCTL -m qos=expedite
Report production host status	PHSTAT	no	MSHOW -a
Report fair-share usage, shares	PSHARE	no	MDIAG -f
Report running-job nodes, pools	SPJSTAT	no	MJSTAT
Unmatched tools:			
Report local job limits	PLIM	no	[NEWS job.lim.*host*]
Report global job limits	BRLIM	no	[none]
Report... host configurations, and "constrainable" features	LRMMGR show default_config show feature *	no	MDIAG -t
Report recent memory use	PHIST	no	[none]
Report job-control efficiency	[none]		SHOWSTATS
Estimate when job will start	[none]		SHOWSTART *jobid*
Report available node count	[none]		SHOWBF

Terminology Comparison

While there is a large vocabulary overlap among different job-control and resource-management systems, some important distinctions have separate names and some common terms (such as 'partition') mean different things in different contexts. These terms appear not only in informal explanations (user documentation) describing tools, but also as labels or headings within tool-generated status reports.

The table below summarizes the standard LCRM working vocabulary and shows for each term the Moab and SLURM equivalent terms (if any).

LCRM Term	Moab Term	SLURM Term	Brief Explanation
user	user	user	Person submitting a batch job (e.g., gsmith)
-	group	group	UNIX group of user (e.g., lcstaff)
node	node or host	node	A single motherboard, usually with multiple CPUs (e.g., atlas75)
host	host	host	One or more nodes managed as a single system (e.g., atlas)
pool	class or queue	partition	A subset of nodes on a host (e.g., pbatch)
partition	partition	-	One (with Moab) or more (with LCRM) hosts managed as a scheduling unit
bank	account	-	The "project" charged for resource use (e.g., science)
job class	QOS (quality of service)	-	Normal, standby, or expedited scheduling status
priority	priority	-	A job's relative ranking to run (0 to 1 for LCRM, -n to +m for Moab)

Moab and LCRM also use (mostly) different terminology to reveal the current state or status of a job being managed by either system. For a comparison of their different approaches to job states and a status-conversion table, see the "Job Status (State) Comparison" section (page 10) below.

Environment Variables in LCRM and Moab

LCRM's Approach:
LCRM uses UNIX environment variables extensively to define the context for your batch job, execute the job, and manage the job's resources (such as the paths searched or work directories used). The "Batch-Job Environment Variables" section (URL: http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3.4) of LC's Environment Variables user manual explains and compares in detail the four distinct sets of such variables involved whenever LCRM runs a job.

In particular, LCRM uses the environment variables listed here to preserve information to pass to the job from the host on which you submitted it:

```
PSUB_DEP_JOBID
PSUB_HOME
PSUB_HOST
PSUB_JOBID
PSUB_LOGNAME
PSUB_PATH
PSUB_REQNAME
PSUB_SUBDIR
PSUB_TZ_ENV
PSUB_USER
SESSARGS
```

If you submit a batch job using the PSUB emulator on an LC machine that has Moab installed in place of LCRM, Moab mimics LCRM and sets these same PSUB environment variables for use by your job when it runs. If you convert (page 16) your job-control script to Moab format, however, and submit it using MSUB, then Moab handles relevant environment variables differently (see the next paragraph).

Moab's Approach:
As currently configured on LC machines, Moab uses *no* environment variables of its own to manage the MSUB batch jobs that it runs. In particular, none of the variables listed above has any role at all for the MSUB-format or "native" jobs that Moab manages. Likewise, the automatically purged temporary directory whose name resides in PCS_TMPDIR for running LCRM jobs does not exist for native Moab jobs.

Instead, Moab relies on environment variables provided by (and assigned values by) each cluster's underlying resource manager (at LC, that is always SLURM (URL: http://www.llnl.gov/LCdocs/slurm)). Moab at LC currently uses the following SLURM-assigned environment variables (as explained in the "Environment Variables" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8) of the SLURM Reference Manual or in SRUN's MAN page) when it runs a batch job:

```
SLURM_CPU_BIND_LIST
SLURM_CPU_BIND_TYPE
SLURM_CPU_BIND_VERBOSE
SLURM_JOBID
SLURM_LOCALID
SLURM_MEM_BIND_LIST
SLURM_MEM_BIND_TYPE
SLURM_MEM_BIND_VERBOSE
SLURM_NNODES
SLURM_NODEID
SLURM_NODELIST
SLURM_NPROCS
SLURM_PRIO_PROCESS  (added just for Moab)
SLURM_PROCID
SLURM_TASKS_PER_NODE
SLURM_TASK_PID  (added just for Moab)
SLURM_UMASK  (added just for Moab)
```

If your Moab-managed job needs any environment variables besides those listed above, you have two choices to supply them:
(a) invoke MSUB's -V (uppercase vee) option to export *all* environment variables from your job's submittal environment to its execution environment, or
(b) invoke MSUB's -v (lowercase vee) option to export only those submittal-environment variables that you specify on the MSUB execute line.

Job Status (State) Comparison

Job states or status values reported for LCRM by PSTAT -A or for Moab by SHOWQ or CHECKJOB are intended to reveal whether or not a submitted job is running, and, if not running then the reason it is not.

While some states or status values are fairly self-explanatory (RUN), others call for "insider" technical information to properly interpret (see the long explanation of the 47 different possible LCRM job states in the "Status Values" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s4.1) of the LCRM Reference Manual, for example). To further complicate this state-interpretation problem, the same waiting job often has a different reported status at the same time depending on whether the job-reporting tool is provided by LCRM (e.g., ELIG), by Moab (e.g., IDLE), or by SLURM (e.g., PENDING).

The first subsection below tells how to discover the state of a Moab-scheduled job, lists the possible Moab state values, and shows where to get further job-state explanations from Moab. The second subsection reveals for each job state reported by LCRM the corresponding status used by Moab (if there is one).

Moab Job States

HOW.

You can discover the state (current status) of a Moab-scheduled job *nnnnn* by using either:

- CHECKJOB *nnnnn*
  and then looking on the report's second row, marked STATE, or
- SHOWQ
  and then looking in the report's third column (headed STATE), where job numbers appear in sequence in the first column.

WHAT.

Possible Moab job states are (in alphabetical order):

```
BatchHold
Completed
Deferred
Idle
NotQueued
Removed
Running
Staging
Starting
Suspended
SystemHold
UserHold
```

WHY.

Moab intends its job states to be "self-explanatory" in the sense that they offer no underlying explanation at all. For example, a job might be IDLE because a higher-priority job is running, because a job on which the IDLE job depends has not yet completed, or because of insufficient required resources. LCRM's many job state names tried to reveal why each job was in its current state. With Moab, you must dig for that explanatory information by running CHECKJOB and studying the last few lines of its long report (the section headed NOTE:, where specific job dependencies and unmet needs are spelled out).

Moab Equivalents of LCRM Job States

LCRM used almost 4 dozen descriptive tags to report the current status of a job and to (also) suggest why the job was in that state (e.g., WAIT to show that a job had not started specifically because its assigned start time had not yet arrived). The "Status Values" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s4.1) of the LCRM Reference Manual explains the hidden meaning of every LCRM job state.

This table shows for each (alphabetical) LCRM job state (status) the corresponding job state assigned by Moab, if there is one. Often multiple LCRM states map into a single Moab state, and some former LCRM states (usually involving limits) are no longer enforced (and so will never appear in PSTAT reports) on Moab-scheduled LC machines.

LCRM Job State	**Moab Job State**
ACCOVER	[no longer enforced]
BAT_WAIT	Staging
CMPLETED	Completed
CPU&TIME	Idle
CPUS>MAX	Idle
DEFERRED	Deferred
DELAYED	[no longer enforced]
DEPEND	Idle
ELIG	Idle
ELIG_SBY	Idle
HELDs	BatchHold, SystemHold
HELDu	UserHold
HLD_IDL	UserHold
HOLDING	SystemHold
JRESLIM	[no longer enforced]
MULTIPLE	Idle
NEW	Staging
NOACCT	[no longer enforced]
NOBANK	[no longer enforced]
NOCONF	SystemHold
NODE>MAX	Idle
NODE<MIN	Idle
NONEW	Idle
NOPRISRV	Idle
NOTIME	[no longer enforced]
NRESLIM	[no longer enforced]
NTRESLIM	[no longer enforced]
PTOOBIG	[no longer enforced]
QCKPLIM	Idle

LCRM Job State	Moab Job State
QTOTLIM	[no longer enforced]
QTOTLIMU	[no longer enforced]
REMOVED	Removed
RES_WAIT	[no longer enforced]
RM_WAIT	Removed
RM_PEND	Removed
RUN	Running
RUN_SBY	Running
STAGING	Staging
TERMINATED	Removed
TOOLONG	[no longer enforced]
TQUOTA	[no longer enforced]
WAIT	Idle
WCPU	Idle
WHOST	NotQueued
WMEM	[no longer enforced]
WMEML	[no longer enforced]
WMEMT	[no longer enforced]
WPRIO	Idle

PSUB Options Conversion

On LC machines where Moab has replaced LCRM, you can still invoke PSUB to submit a job script, and you can still include PSUB options either on the execute line or within the script as #PSUB directives. An emulator or "wrapper script" automatically attempts to convert your PSUB job into an MSUB job that Moab can manage. The first subsection below explains the four categories of PSUB job-control options and tells how each category is handled during the attempted conversion for MSUB. The other subsections then reveal for those PSUB options that have MSUB counterparts what the translation is and what its implications are for how your job will behave.

Summary of Accepted, Ignored, Rejected PSUB Options

LC divides all PSUB job-control options into four groups based on the level of support that they receive from Moab's MSUB tool.

(1) ACCEPTED (Translated):
The PSUB emulator (wrapper script) *accepts* each of these PSUB options, translates it into a functionally equivalent MSUB option, and then submits the job using the option as one of its run specifications. (For examples and explanatory commentary, see the task-oriented (not alphabetical) analysis of accepted options below (page 17).)

PSUB Option	MSUB Counterpart
-A	-a
-b	-A
-c	-l feature=
-d	-l depend=
-e	-e
-eo	-j
-expedite	-l qos=expedite
-H	--help
-i	
-lM	
-ln	-l nodes=
-mb	-m b
-me	-m e
-nokill	-l resfailpolicy=ignore
-nr	-r n
-o	-o
-p	-p
-pool	-q
-prj	-l project=
-r	-N
-s	-S
-standby	-l qos=standby
-tM	-l walltime=
-tW	-l walltime=
-v	
-x	-V

(2) IGNORED (With Warning):

The PSUB wrapper script ignores each of these PSUB options (because they have no MSUB counterpart or because LC has not yet implemented this Moab feature), always issues a warning that this job specification is being ignored, and then submits the job without it.

```
-lc
-ld
-lF
-lf
-lo
-lr
-ls
-lt
```

(3) IGNORED (Without Warning):

The PSUB wrapper script ignores each of these PSUB options (many are seldom needed or obsolete) but it does *not* warn that they have been discarded (unless you explicitly request verbose reporting with -v). The job is submitted without these specifications (although you may reimpose some of them by invoking SLURM's SRUN command inside your job script; see below (page 23)).

```
-cpn
-creds
-dm
-exempt
-g
-ke
-ko
-mn
-nc
-net
-nettype
-nobulkxfer
-np
-re
-ro
```

(4) REJECTED:

The PSUB wrapper script *rejects* each of these PSUB options (as not implemented on current Moab machines at LC or otherwise impossible to support) and then rejects the whole requested job submittal because of them. Moab does not try to run jobs with rejected options.

```
-bgl     [unique to BG/L machines]
```

Script Converter (LCRM2MOAB)

To change an LCRM job-control script (with included #PSUB directives) into a "native Moab" script for use with MSUB, you can log on to any LC machine where Moab has been installed and run the LCRM2MOAB script converter by typing

```
lcrm2moab -i yourpsubscript
```

where LCRM2MOAB

- resides in /usr/bin.
- reports to your terminal (or to a file if you redirect its output for study later) the same warnings about unsupported PSUB options ("option -lt is ignored in Moab") that you would see if the included #PSUB directives were on the PSUB execute line when you ran Moab's PSUB emulator (and helpfully, even options ignored *without* warning when you run PSUB are still reported by LCRM2MOAB, such as -ro).
- generates (in the directory where you run LCRM2MOAB) an output file called jobScript.moab.*nnnnn*
  which contains a (partial) translation of your original PSUB job script into an MSUB job script for your review:
  - ◊ Translatable ("accepted") #PSUB options/directives are *converted* into their correct MSUB counterparts with the original directive preserved as a comment. For example,
    #MSUB -l walltime=40 ##PSUB -tM 40
  - ◊ Untranslatable ("ignored") #PSUB options/directives are *removed* from the output file entirely. No ##PSUB comment marks their former place in the MSUB version of the script.
  - ◊ WARNING: Environment variables (page 8) that exist only for PSUB jobs are *not* converted, flagged, or removed in the MSUB version of the script. For example, the line
    echo jobid = $PSUB_JOBID
    would remain in LCRM2MOAB's output script even though it could not work in a native MSUB job. You must find and convert or remove such imbedded PSUB environment variables yourself.

Basic PSUB/MSUB Options

Moab accepts four kinds of basic PSUB job-sumbittal options and implements them with MSUB counterparts.

JOB DEPENDENCIES:

-A — holds your batch job until the specified date and time of day (in many supported formats).
MSUB uses: -a [[[[CC]YY]MM]DD]hhmm[.SS]
Example: -a 10151400
holds the job until October 15 at 2:00 p.m. (this date string contains no punctuation).

-b — specifies the bank (MSUB calls it an account) from which your job's used resources are drawn.
MSUB uses: -A *accountname*
Example: -A science

-c — specifies the "constraints" or features that a host must have to run your job.
MSUB uses: -l feature=*value*
Example: -l feature=lustre
Run MDIAG -t on any target cluster to discover its current constraints (called "features").

-d — specifies the PSUB jobid of a job that must complete before this job can start.
MSUB uses: -l depend=aftercompletion:*msubjobid*
Example: -l depend=aftercompletion:12345

EXECUTION DETAILS:

-H
(help) lists all PSUB execute-line options and ends.
MSUB uses: --help
This is a syntax reminder only, not an explanatory help message.

-e
specifies a file name (default is *submitdir/yourscript*.e*jobid*) for job error messages.
MSUB uses: -e (default is the same)
Use an absolute pathname with -e to avoid your submittal directory, or use -eo (MSUB -j, see the next item) to put error and log messages into one output file named by -o (a recommended strategy).

-eo
combines ("joins") error and log messages into one output file named by -o.
MSUB uses: -j [oe|n] (argument oe merges output, n does not)
Example: -j oe (the merged file is named by -o)

-mb
sends mail to you (from SLURM) when your job begins.
MSUB uses: -m b
This really invokes the SRUN option --mail-type=begin.

-me
sends mail to you (from SLURM) when your job ends.
MSUB uses: -m e
This really invokes the SRUN option --mail-type=end. Also, native MSUB accepts -m be
to simultaneoulsy request mail when your job both begins and ends.

-nokill
continues to run a job even if a node allocated to the job fails (one job step will likely be lost but subsequent steps may still run if the job has built-in fault tolerance).
MSUB uses: -l resfailpolicy=ignore
Moab treats such fault tolerance as a job "resource."

-nr
prevents rerunning a job (that is, beginning its execution again from step one) rather than just restarting it (continuing on from a checkpoint). Rerunning is the default behavior for both LCRM and Moab.
MSUB uses: -r n
While restarting a job is often desirable, trying to rerun it from the beginning may just waste resources. Hence this is often a default worth overriding with this option.

-o
specifies a file name (default is *submitdir/yourscript*.o*jobid*) for job log messages.
MSUB uses: -o (default is the same)
Example: -o /g/g16/jfk/mylogs/log_abc
To avoid losing your log file in your submittal directory, specify a better-planned location here with an absolute pathname that will help you later.

-prj — specifies a convenient project name (any string up to 127 characters) used in subsequent PSTAT reports about this job.
MSUB uses: -l project=*string*
Example: -l project=bigtest

-r — specifies a job name (up to 15 characters, % is not allowed and the first character cannot be a digit) for use in subsequent reports. The LCRM default job name is the name of the job's script.
MSUB uses: -N *jobname*
Example: -N testrun27a
Moab also uses your job's script name as its default job name.

-s — (lowercase ess) specifies the absolute pathname of the shell that your job should use (the default varies by machine).
MSUB uses: -S *shellpath*
Example: -S /bin/csh
Note that MSUB uses *uppercase* ess for this option. Another way to specify your job's shell is to use the sting #!*shellpath* as the very first executable line within your script (after the #PSUB or #MSUB directives).

-v — (lowercase vee) verbosely reports nonfatal errors detected when you submit your job.
MSUB uses: MSUB automatically reports nonfatal errors during job submittal.
Note that MSUB's native -v (lowercase vee) option has a very different role, to export *specified* environment variables to the job's execution environment. See -x below.

-x — (lowercase eks) exports *all* environment variables from your submittal environment to your job's execution environment.
MSUB uses: -V (uppercase vee)
Note the difference between MSUB's -v (lowercase, described in the preceding item) and -V (uppercase, described here) options.

POLITICAL:

-p *prio*	(for authorized users only) overrides your job's fair-share political priority and forces it to value *prio*. MSUB uses: -p *prio* where *prio* is any integer from -1024 to +1023 (the default is 0). Example: -p 500
-expedite	(for authorized users only) starts your job as soon as possible (PSUB accepts this option from other users but ignores it). MSUB uses: -l qos=expedite MSUB regards this as a "quality of service" job resource.
-standby	(for any user) gives your job such a low priority (STANDBY class) that it runs only if no normal or expedited jobs are available to run. MSUB uses: -l qos=standby MSUB regards this as a "quality of service" job resource.

RESOURCE USE:

-lM — specifies your job's per-node (high-water-mark) memory needs (or "resident set size") as advice for the LCRM scheduler (not an enforced limit).
MSUB uses: -l pmem=*mmmm*[kb|mb|gb]
Currently accepted but ignored, *not* actually used for LC MSUB jobs.
Example: -l pmem=512mb

-pool — specifies a named subset of nodes (MSUB calls this a queue) where your job should run.
MSUB uses: -q *poolname*
Example: -q pdebug

-tM — specifies the maximum CPU time per task (in minutes by default).
MSUB uses: NOT offered by MSUB, see -tW instead.

-tW — specifies the maximum wallclock time (elapsed run time) for your job (in minutes by default).
MSUB uses: -l walltime=*nnn*
Example: -l walltime=3600
WARNING: MSUB uses *seconds* as its default time unit and uses the format HH:MM:SS to change units. Thus

```
    3600 [seconds]
  600:00 [minutes]
10:00:00 [hours]
```

all express the same amount of elapsed time for MSUB.

Parallel PSUB/MSUB Options

The PSUB options most relevant to managing parallel jobs are:

-np	(default was 1, replaced -cpn) formerly specified CPUs per node but now is just advisory. PSUB's -ln now fills this role (see below). MSUB uses: the counterpart option -l ppn=*num* is *not* now implemented for Moab-scheduled clusters at LC. Instead use -l nodes=*num* (see next item) or invoke specific SRUN node-control options within your job's own script.
-ln	requests at least *minnodes* for your job and optionally a node-count range from *minnodes-maxnodes*, perhaps with specified attributes (such as CPUs or memory) per node. MSUB uses: -l nodes=*num* WARNING: attribute support with this -l suboption is *not* currently implemented on LC's Moab-scheduled clusters (so do *not* try -l nodes=32:ppn=4). Instead, invoke SLURM's SRUN tool within your job's script and specify your node attribute needs by using SRUN options directly.

SRUN-Replaced PSUB/MSUB Options

Some PSUB options for which no MSUB counterparts exist (and hence which are ignored, not implemented, by Moab) can nevertheless be implemented on the specific cluster where your job runs by executing SLURM's SRUN utility inside your job-control script and invoking suitable SRUN within-cluster options.

The PSUB options for which this SRUN-replacement strategy is relevant appear to be these:

```
PSUB                  SRUN Alternative

-np (formerly -cpn)   --ntasks (-n)

(AIX only, and Moab is not yet on any AIX LC machine)
-g                    --network (IP | US)
-net                  --network (MPI | LAPI)
-nettype              --network (SN_ALL | SN_SINGLE)
-nobulkxfer           --network (BULK_XFER)
```

Details will go here when available.

SLURM-Moab Interaction

Configuration:

When properly integrated in a production computing environment (such as LC's set of clusters on OCF or SCF), Moab primarily manages batch jobs (among clusters) while SLURM primarily manages compute resources (on each cluster). SLURM fills the role "below Moab" formerly filled by LoadLeveler on IBM AIX machines. Each of these software products must be appropriately configured to interact successfully with the other. Relevant background documentation includes:

- Moab-SLURM Integration Guide (URL: http://wwww.clusterresources.com/products/mwm/docs/slurmintegration.shtml) http://www.clusterresources.com/products/mwm/docs/slurmintegration.shtml is published by Cluster Resources, Inc., and gives four pages of steps to follow and decisions to make for Moab abministrators.
- Moab Cluster Suite Integration Guide (URL: http://www.llnl.gov/linux/slurm/moab.html) http://www.llnl.gov/linux/slurm/moab.html is published by Lawrence Livermore National Laboratory's SLURM project. It gives two pages of preparation and configuration steps to enable a smooth SLURM interface with Moab.

Of course, neither of these general documents for system administrators can reveal for typical users the specific configuration decisions made on LC production machines (nor any between-machine differences here). Disclosing those LC-local configurations is the (future) purpose of this section.

Environment Variables:

Also note that while LCRM deployed quite a number of its own environment variables to manage batch jobs submitted to it, Moab (as currently configured at LC) relies exclusively on SLURM-assigned environment variables for job control (as discussed above (page 8)). Most of these SLURM environment variables (and their equivalent SRUN options) are explained in the "Environment Variables" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8) of LC's SLURM Reference Manual.

SRUN Options:

Some PSUB job-control options have no direct MSUB counterpart, but nevertheless you can use SLURM's SRUN utility inside your job script to achieve similar results. See the section above (page 23) on "SRUN-Replaced PSUB/MSUB Options" for more details (when available).

Job Scheduling

As the "Job Scheduling" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s4.5.2) of the LCRM Reference Manual reveals in detail, LCRM assigns each job a scheduling priority that equals the weighted sum of *three* underlying subpriorities:

- Political priority--
  a "fair share" measure of the computing resources already consumed by the job's user or bank compared with the share of resources that "should have" been consumed. In general, underserved users get priority over those that have recently used many resources. This value is heavily weighted.
- Aging priority--
  a measure of how long a job has waited for resources compared to other competing jobs. This value is moderately weighted.
- Technical priority--
  a measure of how efficiently a job can actually use available computing resources (nodes, memory, run time, etc.) compared to other jobs competing for those same resources. This value is lightly weighted.

Moab offers batch-queue administrators these same three subpriorities as well as several others. Currently, Moab is configured to use these three subpriorities in the same way and to the same extent (weighting) as LCRM does, and *not* to use any additional job subpriorities to actually schedule pending jobs. The current Moab scheduling weights are:

```
Political:        70%
Aging:             0%
Technical:        30%
```

Different weightings, and perhaps even a different mix of scheduling subpriorities, are possible in the future as LC gains more experience deploying Moab with "normal" production workloads.

LIBLRM (Remaining Time) Alternatives for Moab

This section compares the different ways available to a job running under Moab for discovering how much time remains before the job is terminated.

LIBLRM ROLE:

On LCRM-scheduled machines the API library /usr/local/lib/liblrm served two roles--
(1) It provided calls to query a user's banks and charge the calling job to a different bank.
(2) It provided calls to warn a job when its requested and scheduled time limit was about to be reached.
LIBLRM is not available on Moab-scheduled machines, however.

ALTERNATIVES:

Two different strategies allow a job to discover when its time limit is about to be reached (requesting a signal near the end and polling for its remaining time). How best to implement either of these strategies depends on the job's scheduler (Moab or LCRM), on the resource manager (SLURM or not), and on the other places where you also need to run the same application. These charts summarize the alternatives, while the subsections below explain and compare them in detail.

REQUEST A SIGNAL:

For this environment	Use this library	Use this routine
LCRM scheduled	liblrm	lrmsig_register, lrmgettime
Moab emulating LCRM	liblrmemu	lrmsig_register, lrmgettime
Native Moab scheduled	msub -l *signum* or psub -S *signum*	

POLL FOR REMAINING TIME:

For this environment	Use this library	Use this routine
LCRM scheduled	liblrm	lrmwarn
	or	
	libyogrt	yogrt_remaining
Moab emulating LCRM	liblrmemu	lrmwarn
	or	
	libyogrt	yogrt_remaining
Native Moab scheduled		
.....with SLURM	libslurm	slurm_get_rem_time
	or	
	libyogrt	yogrt_remaining
.....without SLURM	Moab API	MCCJobGetRemainingTime
	or	
	libyogrt	yogrt_remaining

LCRM Emulation Library (LIBLRMEMU)

On machines scheduled by Moab instead of LCRM, liblrm is not available. However, LC provides a shared (not static) emulation library called liblrmemu to fill the same role. Jobs can make all the same calls to liblrmemu that they formerly made to liblrm, allowing you to leave your application codes unchanged. The time-remaining calls get time information in a different way that does not depend on LCRM, while the bank calls simply yield no results (details in the subsections below).

LIBLRMEMU Bank Routines

The LCRM library liblrm allowed you to list all banks, your current bank, and your default bank. Jobs could also change their current or default bank while underway. While Moab "accounts" function somewhat like LCRM banks, there are no "current" accounts and jobs cannot dynamically change their default Moab account. Hence, in emulation library liblrmemu these five routines are present for backward compatibility

```
lrmgetallbanks()
lrmgetcurbank()
lrmgetdefbank()

lrmsetcurbank()
lrmsetdefbank()
```

but none of them actually works (they are all "no-ops").

LIBLRMEMU Signal Emulation

Under LCRM, your job could call

```
int lrmsig_register (int sig, long mintime, int *lrmstatp);
```

to request from LCRM a (user-specified) signal when the (user-specified) time remaining had been reached. When that time was reached, LCRM signaled your job, which could then confirm the time remaining by calling either

```
int lrmgettime (long *total, long *used, long *maxtime,
                long *avail, int *lrmstatp);
```

or

```
int lrmgetresource (long *total, long *used, long *maxtime,
                    long *avail, long *stoptime, long *arus,
                    long *maxarus, long *memint,
                    long *maxmemint, int *lrmstatp);
```

After confirmation, your job (typically) saved its data and ended gracefully.

The liblrmemu emulation library supports all three of these signal-management routines for backward compatibility. But liblrmemu, unlike liblrm, populates only the AVAIL and STOPTIME parameters of the latter two routines. Note also that the libslurm library routine slurm_get_rem_time does *not* successfully return a job's grace time on any LCRM-scheduled machine. So libslurm is not a completely portable

substitute for lrmgettime or lrmgetresource even across SLURM-managed clusters at LC. See the "Polling for Time Remaining" section (page 32) below for more details.

LIBLRMEMU Polling Emulation

Instead of registering a signal, the second liblrm grace-time method sets a local flag and passes a reference to this flag along with the desired grace time to the library routine

```
int lrmwarn (int sig, log mintime, int *warn,
             long *stoptime, int *lrmstatp);
```

The job polls this flag regularly, and LCRM changes the flag when the remaining time reaches the requested grace period.

The emulation library liblrmemu supports lrmwarn by forking a new process that periodically invokes the libyogrt library (page 30) routine yogrt_remaining to detect your job's remaining time and set WARN when this drops below MINTIME (it also sets STOPTIME). Because of this internal dependence on the libyogrt library, you may prefer to use that alternative directly in your job (see the next section for details).

LIBYOGRT Remaining Time Library

PORTABILITY:

The LC-developed dynamic libyogrt library provides a simple, fast, portable way for a parallel application to determine how much time remains before it will terminate (the name is an acronym for "your one get remaining time" library). Unlike some of the other methods discussed here, libyogrt works with either the Moab or the LCRM job scheduler and with either the Linux/CHAOS or the AIX operating systems. Each system administrator installs the appropriate version of libyogrt so that once your code calls it you need change nothing for it to get a job's remaining time in a great variety of execution environments. (Note that installation locations vary. For example, libyogrt is hidden in /opt/freeware/lib on AIX machines.)

TIME SOURCES:

The key libyogrt routine, yogrt_remaining, gets your job's remaining run time from
(1) the SLURM underlying resource manager if SLURM is in use, or if not then from
(2) the Moab scheduler, or if not in use then from
(3) the LCRM scheduler.
These three cases cover all possible LC batch-job system configurations. And libyogrt can be easily extended to cover all available ASC tri-lab system configurations as well.

PERFORMANCE:

The routine yogrt_remaining is specifically designed not to undermine the performance of large parallel applications. Once it gets the remaining time from one of the three sources cited above, it maintains an internal cache so that subsequent calls involve only the time() routine and simple arithmetic.

USAGE:

Libyogrt's countdown starts the first time that your job calls the library (but not before that). So you should invoke yogrt_remaining once at the start of your application to begin properly:

```
while [work] {
   if (yogrt_remaining(0) < gracetime) {
       save_state();
       exit(0);
   }
   do_work();
}
```

Yogrt_remaining returns the number of seconds left in your job's wall-time resource allocation. Note that only task0 of a parallel application can call yogrt_remaining successfully (calling it from any other task returns -1).

ENVIRONMENT VARIABLES:

You can change libyogrt's default behavior by setting any of several relevant environment variables (but this is seldom needed or even desirable). For example, if you assign an integer value in seconds to the YOGRT_REMAINING environment variable (such as YOGRT_REMAINING=3600 for 1 hour), then a call to the yogrt_remaining routine will count down from that assigned value instead of from the time returned by the local resource manager or scheduler. The local libyogrt MAN page notes a few other exotic cases.

Native Moab and SLURM Time APIs

If your job will no longer run on any LCRM-scheduled machine, you can abandon backward-compatible ways to get its remaining wall-clock time. Signal support is available directly from Moab, while both Moab and SLURM offer native polling routines. (Of course, as noted in the previous section, libyogrt (page 30) provides automatic LCRM compatibility without sacrificing forward integration with Moab or SLURM to report remaining job time in computing environments where LCRM is absent.)

Requesting a Signal From Moab

For jobs that will never run under LCRM but only under Moab, you can request a signal when a specified grace time is reached when you submit the job to Moab for scheduling. Use either of these job-submittal execute lines (on LC machines):

```
msub -l signumber[@remseconds] ...

psub -S signumber[@remseconds] ...
```

(for example, msub -l 32@120) where

signumber is an integer that specifies which signal your job wants to receive when its grace time is reached (you must provide your own signal handler), and

remseconds is your preferred grace time (the time remaining in seconds until your job will be terminated). Moab's default *remseconds* is 60.

Polling for Time Remaining (Moab, SLURM)

For jobs that will never run under LCRM but only under Moab (or on machines where SLURM manages the underlying compute resources), you can poll for your job's remaining wall time using native SLURM or Moab library routines. The former approach is very accurate, while the latter is accurate to within a minute only.

SLURM Library:
On all SLURM-managed clusters (regardless of operating system), libslurm offers

```
long slurm_get_rem_time (
     uint32_t jobid );
```

which returns the number of seconds remaining before a job (with SLURM *jobid*) exhausts either its own time limit or that of the SLURM partition in which it runs (whichever is smaller). Routine slurm_get_rem_time caches the remaining time locally and only checks the SLURM controller once per minute.
WARNING: On LCRM-scheduled systems where SLURM has replaced LoadLeveler, SLURM is *not* aware of job time limits, so never use routine slurm_get_rem_time on LC machines still scheduled by LCRM.

Moab Library:
Moab (on LC machines and even on other ASC tri-lab machines) provides its own native C-language API that returns a job's remaining time regardless of the underlying resource manager. The routine is

```
MCCJobGetRemainingTime(C,JID,Time,Emsg)
```

where C is the address of the interface handle, JID is the job's Moab ID, Time is the remaining time in seconds, and Emsg is an optional human-readable error message. This approach is not as accurate as using libslurm's slurm_get_rem_time in environments where both are available. For more background on the Moab C-langauge API, consult this vendor website:

http://www.clusterresources.com/products/mwm/moabdocs/a.hinterfacing.shtml

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 35).

Keyword	Description
entire	This entire document.
title	The name of this document.
scope	Topics covered in this document.
availability	Where Moab runs.
who	Who to contact for assistance.
introduction	Moab's relation to LCRM and SLURM.
lcrm	Moab counterpart for LCRM features, tools.
tool-comparison	Moab and LCRM job-control tools compared.
terminology	Moab, LCRM, SLURM vocabulary differences.
environment-variables	Different env var treatments in Moab, LCRM.
job-status	Moab and LCRM job states compared.
job-state	Moab and LCRM job states compared.
job-state-moab	Possible Moab job states.
job-state-lcrm	Moab equivalents for LCRM job states.
psub-options	How Moab (MSUB) handles PSUB options.
psub-options-summary	Accepted, ignored, rejected PSUB options.
lcrm2moab	Using LCRM2MOAB script-conversion tool.
basic-options	How MSUB implements basic PSUB options.
parallel-options	How MSUB implements parallel PSUB options.
srun-replaced	How SRUN options replace some PSUB options.
slurm	How Moab and SLURM interact.
scheduling	How Moab schedules batch jobs.
get-time-libraries	Get remaining time without LIBLRM.
liblrm-replacements	Get remaining time without LIBLRM.
liblrmemu	LCRM emulation library for Moab.
liblrmemu-bank	LIBLRMEMU bank routines.
liblrmemu-signal	LIBLRMEMU signal emulation.
liblrmemu-poll	LIBLRMEMU polling emulation.
libyogrt	LIBYOGRT to get remaining time.
native-time	Native Moab, SLURM time APIs.
native-time-signal	Time signals from Moab.
native-time-poll	Time polling with Moab, SLURM.
index	The structural index of keywords.
a	The alphabetical index of keywords.
date	The latest changes to this document.
revisions	The complete revision history.

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
availability	Where Moab runs.
basic-options	How MSUB implements basic PSUB options.
date	The latest changes to this document.
entire	This entire document.
environment-variables	Different env var treatments in Moab, LCRM.
get-time-libraries	Get remaining time without LIBLRM.
index	The structural index of keywords.
introduction	Moab's relation to LCRM and SLURM.
job-state	Moab and LCRM job states compared.
job-state-lcrm	Moab equivalents for LCRM job states.
job-state-moab	Possible Moab job states.
job-status	Moab and LCRM job states compared.
lcrm	Moab counterpart for LCRM features, tools.
lcrm2moab	Using LCRM2MOAB script-conversion tool.
liblrm-replacements	Get remaining time without LIBLRM.
liblrmemu	LCRM emulation library for Moab.
liblrmemu-bank	LIBLRMEMU bank routines.
liblrmemu-poll	LIBLRMEMU polling emulation.
liblrmemu-signal	LIBLRMEMU signal emulation.
libyogrt	LIBYOGRT to get remaining time.
native-time	Native Moab, SLURM time APIs.
native-time-poll	Time polling with Moab, SLURM.
native-time-signal	Time signals from Moab.
parallel-options	How MSUB implements parallel PSUB options.
psub-options	How Moab (MSUB) handles PSUB options.
psub-options-summary	Accepted, ignored, rejected PSUB options.
revisions	The complete revision history.
scheduling	How Moab schedules batch jobs.
scope	Topics covered in this document.
slurm	How Moab and SLURM interact.
srun-replaced	How SRUN options replace some PSUB options.
terminology	Moab, LCRM, SLURM vocabulary differences.
title	The name of this document.
tool-comparison	Moab and LCRM job-control tools compared.
who	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
17Jul07	job-state-moab	Possible Moab job states added.
	job-state-lcrm	LCRM/Moab state conversion added.
	index	New keywords for new subsections.
18Jun07	environment-variables	MSUB's -V and -v roles contrasted.
	basic-options	-l dependency corrected to -l depend.
	libyogrt	Restriction to task0 noted.
16May07	liblrm-replacements	New major section on ways to "get remaining time" for a job.
	basic-options	16 more PSUB/MSUB option comparisons.
	parallel-options	Two comparisons elaborated.
	introduction	Cross ref to new subsections added.
	index	New keywords for new subsections.
26Mar07	tool-comparison	PHIST, SHOWBF added to chart.
	environment-variables	Cross ref, Moab roles added.
	srun-replaced	-n same as --ntasks.
07Mar07	entire	First edition of Moab manual.

TRG (17Jul07)

UCRL-WEB-228716

TRG (17Jul07) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov